D.3 A conflict at commands implies a conflict at paths

We prove this theorem for two replicas.

We prove that if we have filesystems $O, A, B \neq \perp$ and we have the minimal sequences S_A and S_B and the dirty-sets $dirty_A$ and $dirty_B$ from the update-detectors, then if we have conflicting commands in the sequences, we have dirty-conflicts too.

The fact that we gained the sequences from update-detectors allows us to assume that there are no superfluous commands in the sequences, e.g., a command which leaves the filesystem in the same state.

Since we cannot distinguish between directories by their contents in the model of the paper of Balasubramaniam and Pierce, we have some preconditions.

Our theorem is true if:

- commands do not modify directories to directories (that is, $O(\pi) = X_{Dir} \implies edit(\pi, Y_{Dir}) \notin S_A$ or S_B);
- all directories have the same contents (that is, for any F and G, if $F(\pi)$ is a directory and $G(\gamma)$ is a directory then $F(\pi) = G(\gamma)$); and
- there are no superfluous commands in S_A and S_B.

First of all, notice that $C_A(\pi) \leftrightarrow C_B(\gamma)$ can be true only if $\pi \preceq \gamma$ or $\gamma \preceq \pi$. (Otherwise they would commute.) Without loss of generality suppose the first one is true. Now we know: $C_A(\pi) \leftrightarrow C_B(\pi_{esub})$, where $\pi_{esub} = \gamma$.

Our theorem is that

$$\textbf{if } C_A(\pi) \leftrightarrow C_B(\pi_{esub}) \textbf{ then } conflict(\pi),$$

that is, there is a dirty-conflict at path π.

Proof. We know that $C_A(\pi) \in S_A$; therefore $O(\pi) \neq S_A O(\pi)$, where O is the original filesystem (remember that we have no superfluous commands). Hence $\pi \in dirty_A$ according to the definition of this set.

The same holds for $C_B(\pi_{esub})$; therefore $\pi_{esub} \in dirty_B$. This implies that $\pi \in dirty_B$ since $dirty_B$ is up-closed.

Now we need to show that $A(\pi)$ or $B(\pi)$ is not a directory. Suppose that both of them are directories. That is, $A(\pi) = S_A O(\pi) = X_{Dir}$. In other words ($S_A$ is minimal) $C_A(\pi)O(\pi) = X_{Dir}$ since there are no other command on this path. Thus $C_A(\pi)$ is $create(\pi, X_{Dir})$ or $edit(\pi, X_{Dir})$, but we can be sure that $O(\pi) \neq X_{Dir}$ because otherwise we would edit (modify) a directory to a directory, and by assumption we cannot do that.

We also know that $B(\pi) = S_B O(\pi) = X_{Dir}$ since we do not distinguish between directories. Since we know that $O(\pi) \neq X_{Dir}$, we must have a command on π in S_B, namely $C_B^*(\pi)$. This command can be either $create(\pi, X_{Dir})$ or $edit(\pi, X_{Dir})$, but it is equivalent to $C_A(\pi)$ since we must use *create* in both cases if $O(\pi) = \perp$ and *edit* if $O(\pi) = X_{File}$. But then $C_A(\pi) \in S_B$ since $C_A(\pi) \equiv C_B^*(\pi) \in S_B$ which contradicts the definition of conflicting commands.

We also need to show that $A(\pi) \neq B(\pi)$. If so, similarly to the above, $C_A(\pi) \equiv C_B^*(\pi)$ would hold.

Now we know that every condition of a dirty-conflict holds. Our theorem is proved.

E Implementation

We created an implementation based on the algorithms described in Section 5 and Appendix C. Its main purpose was to verify that the algorithms are implementable and they work as we expect them. The program is written in Perl and it runs under UNIX systems. It handles two replicas and does not modify the filesystem; it only detects updates and conflicts among commands. Then it provides the sequences of commands which should be applied to the replicas.

It has no archives of the filesystem; therefore, we must also provide it the original version of the filesystem. This is because the implementation was built to examine the algorithms, not to make a complete synchronizer. It also does not simplify the outcoming sequences with *move* commands or commands on subtrees.

When referring to the contents of directories, it looks at the writable flags of a directory. That is, the contents of two directories are different if one of them is writable for the program and the other one is not.

The brief description of its method is the following:

1. Creates flat representations of the original filesystem and the replicas $O(), A(), B()$.
2. Update-detection: defines the minimal sequences S_A, S_B for which $S_A O = A$ and $S_B O = B$.
3. Orders these sequences using the method discussed in section C.3.
4. Reconciling: detects conflicting commands in sequences.

5. Provides sequences S_A^* and S_B^* for reconciliation.
6. Provides a list of paths where conflicts occurred.

E.1 Equivalence of *edit* and *create* commands

To be able to determine whether two *edit* or *create* commands are equivalent, we introduced a *list of samples* in the program, which is a list of paths. When the program detects an $edit(\pi, X)$ or a $create(\pi, X)$ command as an update, it looks for a path γ in the list for which $A(\pi) = A(\gamma)$ (in case it is detecting updates at replica A). If it finds such a γ, it will attach its number in the list to the command. If not, it appends π to the list and attaches the new number.

That way, two *edit* or *create* commands are equivalent if and only if they are applied to the same path and they have the same sample-number.

E.2 Notation

The program uses the ordinary `directory/directory/file` notation for paths. For commands, it uses:

`EF0/path`	for	$edit(path, X_{File})$;
`ED0/path`	for	$edit(path, X_{Dir})$;
`CF0/path`	for	$create(path, X_{File})$;
`CD0/path`	for	$create(path, X_{Dir})$;
`RM/path`	for	$remove(path)$,

where `0` is the sample-number for which $A(Samplelist(0)) = A(\texttt{path})$.

When marking conflicting commands, it uses '`<->A3`' if the command conflicts with the 3^{rd} command in sequence S_A and '`==>A3`' if the command $S_A[3]$ must precede the current command, but it already conflicts, so the current command cannot be applied.

E.3 Examples

In this section we provide some examples of how the program runs. In the first section of the output the program lists the flat representation of the filesystems (the original and the two replicas). The next section is the result of the detection of updates and conflicting commands. It lists the samples and the sequences of commands (S_A and S_B). Then sequences S_A^* and S_B^* follow which should be applied to replicas to propagate nonconflicting commands. In the last section it lists paths where there were conflicts.

E.3.1 Example 1

This example is the case we discussed in Section D.2.

```
File Synchronizer
  by Elod Csirmaz 21.07.2000
#==List of filesystems============
---Original Filesystem------------
FS-0/dir/
FS-0/dir/file
---Replica A----------------------
---Replica B----------------------
FS-B/dir/

#==Update-detection and conflicts=
---Edit Samples-------------------
---Sequence 0->A------------------
0:RM/dir/file
1:RM/dir/
---Sequence 0->B------------------
0:RM/dir/file

#==Reconciliation=================
---Sequence A->0'-----------------
---Sequence B->0'-----------------
1:RM/dir/

#==Paths of conflicting commands==
==================================
```

As we can see, a situation like this does not cause conflicting updates.

E.3.2 Example 2

This example shows a case when we have conflicts because a conflicting command must precede another. `/dir/file` was modified in replica A.

```
File Synchronizer
  by Elod Csirmaz 21.07.2000
#==List of filesystems============
---Original Filesystem------------
FS-0/dir/
FS-0/dir/file
---Replica A----------------------
FS-A/dir/
FS-A/dir/file
---Replica B----------------------


#==Update-detection and conflicts=
---Edit Samples-------------------
0:FS-A/dir/file
---Sequence 0->A------------------
```

```
0:EF0/dir/file <->B0
---Sequence 0->B-----------------
0:RM/dir/file <->A0
1:RM/dir/ ==>B0

#==Reconciliation================
---Sequence A->0'----------------
---Sequence B->0'----------------
#==Paths of conflicting commands==
dir/
dir/file
================================
```

Actually, in this case the synchronizer cannot do anything, because all commands conflicted.

F Soundness proofs for individual laws

In this section we provide examples of how algebraic laws can be proved. The method we use is to consider as many cases as necessary to be able to predict the result of each command in the law. For example, if the law contains paths π and π_{sub}, then we will consider 21 cases; see below. (Note that we listed all cases considering how π_{sub} can relate to π and if they have children or a parent. Also, we considered the case when $F(\pi)$ has only one child and it is $F(\pi_{sub})$ since in that case, modifying $F(\pi_{sub})$ can make $F(\pi)$ childless.)

If the law contains an *edit* command, we also distinguish between files and directories. If we determine the result of the sequence of commands of the left side in each case and this is the same as that of the right side, then the law holds.

Now we provide an example proof on law 18. We will investigate only the left side of the law, since the right side always gives the broken filesystem.

Case 0
$F(parent(\pi)) = \perp$
$F(\pi) = \perp$
$childless(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 1
$F(parent(\pi)) = A_{File}$
$F(\pi) = \perp$
$childless(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 2
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = \perp$
$childless(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 3
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = \perp$
$childless(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 4
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{File}$
$childless(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 5
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{File}$
$childless(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 6
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$childless(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 7
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$childless(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 8
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 9
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{File}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 10
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 11
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = x_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 12
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = \perp$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 13
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{File}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 14
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = \perp$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 15
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = o_{File}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 16
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = o_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 17
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = o_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$
$F(parent(\pi_{sub})) = A_{Dir}, parent(\pi_{sub}) \neq \pi$
$F(\pi_{sub}) = x_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 18
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{File}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 19
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = x_{Dir}$
$childless(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 20
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$
$F(\pi_{sub}) = o_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$:
$F(parent(\pi)) = A_{Dir}$
$F(\pi) = o_{Dir}$
$children(\pi)$, only one
$parent(\pi_{sub}) = \pi$

$F(\pi_{sub}) = x_{Dir}$
$children(\pi_{sub})$
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Case 21
UNDEFINED
After $edit(\pi_{sub}, dir)$:
UNDEFINED
After $edit(\pi_{sub}, dir)$ and $remove(\pi)$:
UNDEFINED

Preface

The *Research Science Institute Student Reports 2000* exhibits the work of seventy academically talented high school students who participated in the 2000 Research Science Institute (RSI) sponsored by the Center for Excellence in Education in collaboration with the Massachusetts Institute of Technology in Cambridge, Massachusetts, from 25 June through 5 August.

This unique six-week summer program combines theoretical classroom instruction in mathematics and science with one-on-one research experience with scientists from the Massachusetts Institute of Technology, Harvard University, Boston University Medical School, private corporations, and research organizations in and around Boston, Massachusetts. It is the only U.S. program of its kind offered at no cost to participants. The scientists and their respective organizations are acknowledged in the headings of the reports.

These reports are the culmination of students' research in mathematics, physics, chemistry, biology, computer science, and engineering. The faculty of the Institute selected five reports for outstanding writing awards and five for outstanding oral presentation awards. The winning written reports are printed first in their entirety, followed by the abstracts for the reports that won the oral-presentation awards, and finally by the remaining student abstracts.

The seventy students represented thirty-three states of the Union, as well as the countries of Bulgaria, France, Germany, Greece, Hungary, Lebanon, Poland, Serbia, Russia, Singapore, and the United Kingdom. The students were all selected having completed coursework comparable to the third year of high school. The U.S. students averaged 78 of 80 points on the mathematics section and 75 of 80 points on the verbal sections of the Preliminary Scholastic Aptitude Test (PSAT), representing the 99th percentile of all high-school students taking the PSAT.

The unique merits, both academic and social, of the Institute are perhaps best indicated by the high opinions of those who have been affiliated with the program, of whose sentiment the following comments from participants of the 2000 Institute are representative:

> *RSI has allowed me to appreciate that the advancements in science are continuous and extremely rapid.*
>
> *I felt really connected to people and felt that everyone helped reaffirm my love of the sciences.*
>
> *I hope to see these people in college and keep in touch with them for a long, long time.*
>
> *I am much more aware of the constantly expanding fields of math and science. What a happy world to live in!*
>
> *Not only did I learn a lot at mentorship; I discovered a great group of friends.*
>
> *Thank you for encouraging the educational advancement and achievement of students around the world.*

The editor is honored to have taken part in the Institute, once as a participant, once as a staff member, and once as an unofficial volunteer, and is pleased to present to you the excellent work of his successors.

Christopher C. Mihelich, RSI 1997, *Editor*

Table of Contents